3Г

УДК 621.39, 004.7

К.Д. Гуляев, Д.А. Зайцев
Одесская национальная академия связи им. А.С. Попова, Украина
k.guliaiev@gmail.com, zsoftua@yahoo.com

Экспериментальная реализация стека сетевых протоколов E6 в ядре ОС Linux

Представлена первая реализация нового стека сетевых протоколов E6 в ядре операционной системы. Стек E6 разработан для повышения эффективности работы сетей, всецело построенных на основе технологии Ethernet. Он использует единый иерархический E6 адрес на всех уровнях эталонной модели взаимодействия открытых систем и аннулирует протоколы TCP и IP. Экспериментальная реализация добавляет новый системный вызов ядра Linux и новый тип Ethernet E6 кадра. Все стандарты прикладных интерфейсов сохранены в соответствии с RFC за исключением использования E6 адреса вместо IP-адреса, а также вместо Ethernet MAC-адреса.

Повышение эффективности сетевых технологий представляет собой важную научную проблему, одним из возможных решений которой является разработка новых стеков протоколов. Стек протоколов E6 был представлен в [1], затем был получен соответствующий патент Украины на полезную модель [2] и работоспособность E6 сетей была подтверждена путем моделирования [3] в системе CPN Tools [4]. Однако до последнего времени стек E6 не был реализован в среде реальных операционных систем.

Целью работы является разработка структур данных и алгоритмов для программной реализации стека E6, а также выбор операционной системы и интеграция программного обеспечения в соответствующую операционную среду.

Операционные системы Linux имеют открытый исходный код и содержат инструментальные средства для реализации новых стеков протоколов, что обусловило их выбор в качестве операционной среды. Основным принципом реализации был избран подход сохранения прикладных интерфейсов в соответствии со стандартами TCP и UDP [5], [6] и канального интерфейса в соответствии со стандартами Ethernet [7], а также обеспечение независимой работы стека E6 среди других протоколов.

Настоящая экспериментальная реализация не использует средства Ethernet LLC2 и обеспечивает прикладной интерфейс протокола UPD. Реализация прикладных интерфейсов TCP требует использования процедур скользящего окна Ethernet LLC2 вместо соответствующих процедур протокола TCP, что является предметом будущих исследований.

1. Обзор технологии E6

Технология E6 [1], [2] базируется на двух основных идеях: использовать один и тот же единый адрес E6 с длиной в 6 байтов (рис. 1) на всех уровнях эталонной модели взаимодействия открытых систем и использовать процедуры скользящего окна Ethernet LLC2 вместо протокола TCP для гарантированной доставки информации.

```
 0                   1                   2                   3                   4
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
|      E6 Network Address          |                 E6 Host Address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\                                 /
 ----------------\/------------------
           Subnet Mask (as the number of bits of E6 network address)
```

Рисунок 1 – Формат Е6 адреса

В результате только лишь пара номеров программных портов остается для размещения в заголовке пакета (E6p2 заголовке – рис. 2) и остаток работы передается для исполнения программным обеспечением Согласование-Е6 на канальный уровень Ethernet (рис. 3). Что касается дополнительных параметров качества обслуживания ToS, то они могут быть размещены в VLAN заголовке кадра. В действительности Е6 аннулирует протоколы TCP, UDP и IP, а остатком являются 4 байта E6p2 заголовка (экономится не менее 36 байтов заголовков на каждом пакете). Стек Е6 также аннулирует отображение IP-адресов в МАС-адреса и соответствующие протоколы ARP/RARP, которые потребляют достаточно много времени работы хостов и маршрутизаторов.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Рисунок 2 – Заголовок E6p2

Так как Е6 адрес является иерархическим точно так же, как CIDR IP-адрес, он обеспечивает построение глобальных (всемирных) сетей за счет агрегирования адресов хостов и подсетей под общей маской, что существенно сокращает размер адресной таблицы. Технология Ethernet страдает от своих плоских МАС-адресов, которые переполняют адресные таблицы коммутаторов из-за того, что должен быть указан каждый индивидуальный адрес. Так как Е6 адрес имеет такую же самую длину, как Ethernet МАС-адрес (а именно 6 октетов), он может быть непосредственно размещен в поле адреса кадра Ethernet вместо МАС-адреса. Современные Ethernet-адаптеры позволяют присваивать новые МАС-адреса интерфейсам, и присваивается соответствующий Е6 адрес.

Для обеспечения независимого существования технологии Е6 введен новый тип Е6 Ethernet кадра (0xE600). Хосты, поддерживающие стек Е6, принимают Е6 кадры, в то время как другие хосты просто игнорируют их. Обычные Ethernet-коммутаторы доставляют Е6 кадры беспрепятственно, поскольку они обычно не анализируют поле типа кадра (если не установлены специальные фильтры). Традиционные IP-маршрутизаторы теряют неизвестные им Е6 пакеты, случайно доставленные в результате широковещания. Таким образом Е6 может существовать в границах коммутируемой сети Ethernet.

Для обеспечения масштабируемости необходимо разработать специальные коммутирующие маршрутизаторы Е6 (КМЕ6), либо реализовать их как патчи к обычному программному обеспечению известных маршрутизаторов (например, CISCO IOS). Достаточно простым решением по созданию КМЕ6 является их реализация на основе обычного Unix-компьютера с несколькими картами (адаптерами) Ethernet.

```
+-------------+---------------------+---------------------+
|   OSI-ISO   |       TCP/IP        |         E6          |
+-------------+---------------------+---------------------+
| Прикладной  | HTTP, SMTP, VoIP ...| HTTP, SMTP, VoIP ...|
+-------------+----------+----------+---------------------+
| Сеансовый   |          |          |                     |
+-------------+   TCP    +----------+                     |
| Транспортный|          |   UDP    |   E6 Согласование   |
+-------------+----------+----------+                     |
| Сетевой     |         IP          |                     |
+-------------+---------------------+---------------------+
| Канальный   |      Ethernet       |     E6 Ethernet     |
+-------------+---------------------+---------------------+
```

Рисунок 3 – Стек протоколов Е6

Схема доставки пакета, представленная на рис. 4, иллюстрирует преимущества технологии Е6: одна и та же пара Е6 адресов отправителя и получателя остается неизменной на всем пути доставки пакета. КМЕ6 только анализируют Е6 адрес получателя и перенаправляют пакет в порт назначения; отсутствует дополнительное отображение адресов (в отличие от отображения IP-MAC); отсутствуют более 40 октетов заголовков протоколов TCP и IP для каждого передаваемого пакета. Вычислительные ресурсы устройств сохранены для обеспечения более высокой производительности и лучшего качества обслуживания (QoS), что особенно важно в приложениях телефонии (VoIP).

Заметим, что технология Е6 оправдана настоящим состоянием телекоммуникаций и тенденциями их развития – Ethernet доминирует на канальном уровне сетей: локальные сети – 1, 10 Гбит/с, кампус и метрополитен сети – 10 Гбит/с поверх DWDM, магистрали – Carrier-Ethernet и мосты провайдера PBB, сети доступа – Ethernet последней мили, беспроводные сети – радио Ethernet (WiFi).

```
+------------+         +-------+                              +------------+
| E6 Host X  |         | E6SR1 |              E6SR2...E6SRk   | E6 Host Y  |
|            |         |       |                              |            |
|(address E6X)|        | E6Y-> |                              |(address E6Y)|
|            |         | PortB |                   . . .      |            |
+------------+         +-------+                              +------------+
  |                PortA ^    | PortB                  ^     |               ^
  |                      |    |                        |     |               |
  |   +-----+-----+---+  |    |   +-----+-----+---+    |     |   +-----+-----+---+  |
  +-->| E6Y | E6X |...|--+    +-->| E6Y | E6X |...|--+       +-->| E6Y | E6X |...|--+
      +-----+-----+---+           +-----+-----+---+              +-----+-----+---+

      E6 Ethernet frame (unchanged)
```

Рисунок 4 – Схема доставки Е6 пакета

2. Прикладные интерфейсы

Программное обеспечение разработано как загружаемый модуль E6udpmod.ko ядра Linux за исключением незначительных изменений, внесенных в статическую часть ядра с целью добавления нового системного вызова. Условный (conditional) системный вызов 324 с именем E6_call добавлен в статическую часть ядра. Этот дополнительный системный вызов предназначен для реализации всех настоящих и будущих процедур прикладного интерфейса стека Е6. Статическая часть ядра содержит указатель для нового системного вызова 324 (который не задействован в версии ядра 2.6.22.9) на фиктивную процедуру sys_E6call, которая проверяет указатель new_sys_E6call на настоящую процедуру и вызывает ее, в случае если указатель ненулевой (не равен

NULL). Указатель инициализируется значением NULL, так что, когда модуль E6updmod.ko не загружен, ничего не происходит, за исключением того, что sys_E6call может выводить сообщения в системный журнал для индикации вызова. Указатель new_sys_E6call экспортируется ядром для использования в загружаемых модулях. Программный код выглядит следующим образом:

```
/* file syscall_table.S */
ENTRY(sys_call_table)
....
        .long sys_e6call                /* new e6 syscall 325 */

/* file socket.c  e6 new */
long (*new_sys_e6call)(int call, unsigned long __user *args) = NULL;
asmlinkage long sys_e6call(int call, unsigned long __user *args)
{
  int err;
  if( new_sys_e6call != NULL )
    err = (*new_sys_e6call)( call, args );
  else
    { printk(KERN_INFO"*** sys_e6call echo: %d\n", call); err=0; }
  return err;
}
EXPORT_SYMBOL(new_sys_e6call);
/* e6 new end */
```

Оставшуюся часть работы выполняет модуль E6udpmod.ko. При инициализации он устанавливает указатель new_sys_E6call на действительную точку входа обработчика системных вызовов E6, а также находит Ethernet-устройство E6_dev среди зарегистрированных устройств ядра, копирует его аппаратный адрес и регистрирует обработчик новых E6 пакетов:

```
static int e6udpmod_init_module(void)
{
        int err=0;

        printk(KERN_INFO"*** e6udpmod: init devname=%s ***\n", e6_devname);
        /* find e6 device */
        e6_dev = dev_get_by_name(e6_devname);
        memcpy( e6_myaddr, e6_dev->dev_addr, e6_dev->addr_len );
        /* set new e6call function */
        new_sys_e6call = mod_sys_e6call;
        /* register e6 packet handler */
        dev_add_pack(&e6_packet_type);

      return 0;
}
```

После оператора new_sys_E6call = mod_sys_E6call все системные вызовы E6 обрабатываются подпрограммой mod_sys_E6call, которая расположена в модуле E6updmod.ko и работает как диспетчер условных системных вызовов, распознаваемых по их номерам, заданным переменной call:

```
extern long (*new_sys_e6call)(int call, unsigned long __user *args);

long mod_sys_e6call(int call, unsigned long __user *args)
{
  unsigned long a[1];
```

```
  unsigned long a0;
  unsigned char nargs = (1)*sizeof( unsigned long );
  int err;

  printk(KERN_INFO"*** e6udpmod: mod_sys_e6call %d\n", call );

  if (copy_from_user(a, args, nargs))
         return E6ERR_COPY;

  a0 = a[0];

  switch( call ){
  case E6_CALL_PUTMSG:
                err = e6_putmsg( (struct e6msg_buf __user *)a0 );
                break;
  case E6_CALL_GETMSG:
                err = e6_getmsg( (struct e6msg_buf __user *)a0 );
                break;
...
  default:
                err = E6ERR_CALLNUM;
                break;
  }

  return err;
}
```

Интерфейс на прикладном уровне обеспечивается библиотекой E6udplib, которая содержит функции конечного пользователя E6sendmsg, E6rcvmsg, E6regport, E6unregport, E6waitmsg, которые издают в результате своей работы системный вызов 324, используя процедуру syscall из стандартной библиотеки libc:

```
#define SYS_e6call          324
#define E6_CALL_PUTMSG       1
#define E6_CALL_GETMSG       2
...
#define MAXE6BUFSIZE   1496
#define E6ADDRLEN            6

struct e6msg_buf {
        u8 e6dst_addr[E6ADDRLEN];
        u8 e6src_addr[E6ADDRLEN];
        u16 e6dst_port;
        u16 e6src_port;
        u16 e6data_len;
        u8 * e6data;
};

struct e6msg_buf buf;

int e6sendmsg( struct e6msg_buf * b, int * err )
{
 int rc;
 unsigned long a[1];

 a[0]=(unsigned long)b;

 rc = syscall( SYS_e6call, E6_CALL_PUTMSG, a );
```

```
 *err = errno;

 return rc;
}
...
```

Множество программ E6sendmsg, E6rcvmsg, E6regport, E6unregport, E6waitmsg полностью удовлетворяет требованиям RFC 768 к прикладным интерфейсам протокола UDP. Взаимодействие частей программного кода, размещенных в статической части ядра, в загружаемом модуле и в пользовательской библиотеке, проиллюстрировано на рис. 5.

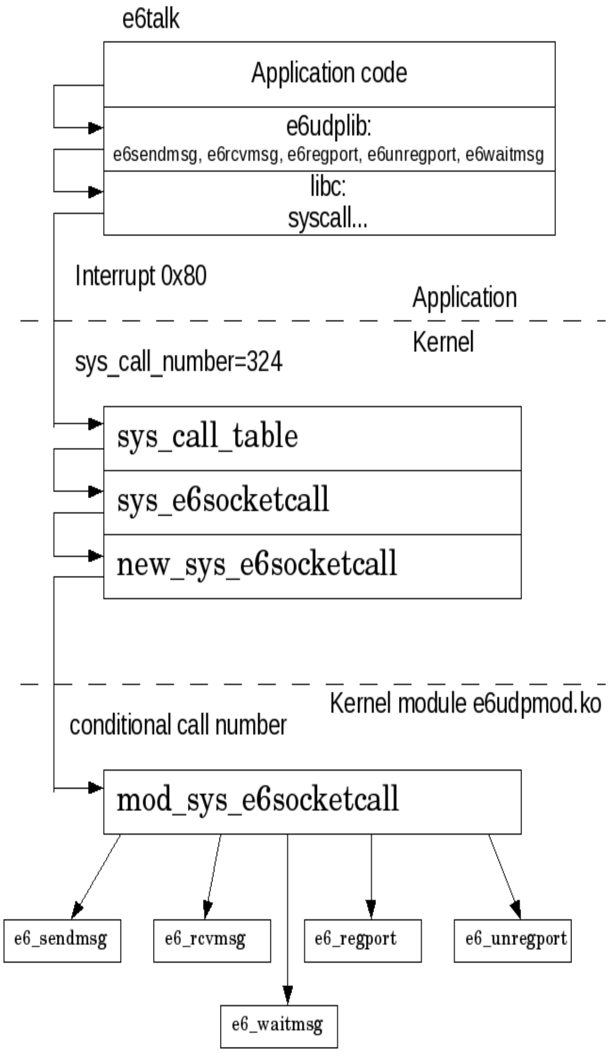


Рисунок 5 – Взаимодействие между программным кодом пользователя, ядра и модуля

Библиотека интерфейса пользователя E6udplib задействована для разработки простого приложения E6talk, которое обеспечивает обмен текстовыми сообщениями в пределах E6 сети, построенной на основе коммутируемой Ethernet. Указанное приложение выполняет реальный обмен информацией, что подтверждает работоспособность стека протоколов E6. Для подтверждения эффективности технологии E6 по отношению к семейству протоколов TCP/IP необходимо еще проделать достаточно большую работу.

Анализаторы трафика, такие как Wireshark, позволяют наблюдать E6 Ethernet-кадры, которые свободно передаются среди других типов кадров и не препятствуют работе других протоколов. Таким образом, E6 сети существуют в параллельном мире по отношению к TCP/IP до тех пор, пока не будут разработаны шлюзы между E6 и TCP/IP сетям, что является направлением будущих работ.

3. Интерфейсы канального уровня

Интерфейс с оборудованием Ethernet обеспечивается в среде ядра Linux структурой struct net_device, которая содержит описание устройства и его функций. Допустимые функции заданы набором указателей, которые указывают на действительные программы текущего драйвера Ethernet; основными программами являются следующие: open, stop, hard_start_xmit. Структура net_device содержит также заголовок очереди выходных пакетов; пакет представлен описанием struct sk_buff *skb.

Интерфейсы канального уровня оставлены без изменений; они лишь только использованы для передачи E6 Ethernet-кадров. Мультиплексирование при отправлении E6 пакета осуществляется посредством использования нового типа E6 Ethernet-кадров:

```
#define ETH_P_E6        0xE600
```

Модуль заполняет sk_buff данными пользователя и E6 заголовками, используя тип кадра ETH_P_E6, и вызывает программу hard_start_xmit, передавая управление драйверу Ethernet для передачи пакета через среду.

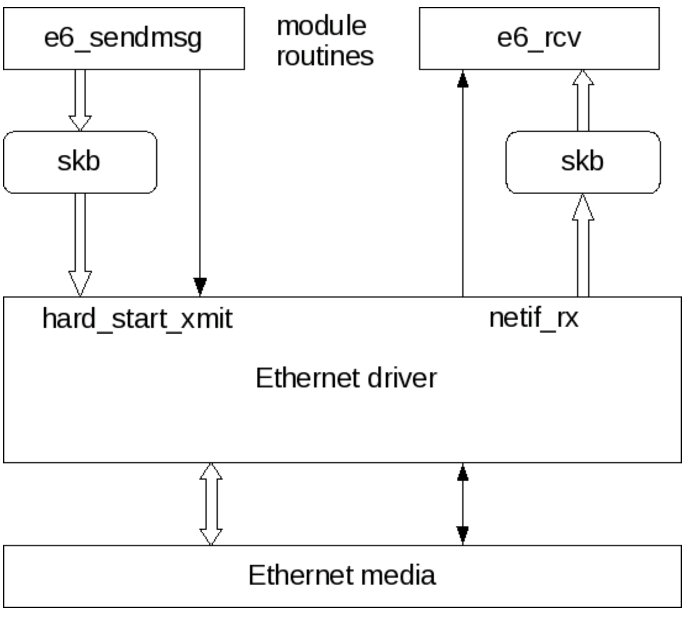


Рисунок 6 – Интерфейсы с канальным уровнем

Получение E6 пакета является более сложным, так как оно выполняется драйвером Ethernet по аппаратному прерыванию Ethernet-адаптера (карты). Драйвер выделяет память и заполняет sk_buff данными принятого кадра и затем выполняет процедуру демультиплексирования, основанную на множестве зарегистрированных типов пакетов. Регистрация нового типа пакета осуществляется с помощью структуры struct packet_type; структура инициирована следующим образом:

```
static struct packet_type e6_packet_type = {
	.type = __constant_htons(ETH_P_E6),
	.func = e6_rcv,
	.gso_send_check = NULL, /*e6_gso_send_check,*/
	.gso_segment = NULL, /*e6_gso_segment,*/
};
```

Таким образом, после следующего оператора в указанной выше процедуре инициализации модуля (E6udpmod_init_module)

```
dev_add_pack(&e6_packet_type);
```

все принятые пакеты типа ETH_P_E6 обрабатываются программой E6_rcv, расположенной внутри модуля E6udpmod.ko. Следовательно, установлены двусторонние интерфейсы с канальным уровнем: для отправления E6 пакетов и для получения E6 пакетов. Заметим, что E6 пакеты и соответствующие E6 кадры передаются независимо среди других типов кадров Ethernet. Схема, представленная на рис. 6, поясняет интерфейсы модуля E6udpmod.ko с канальным уровнем Ethernet.

4. Внутренние структуры данных и программы

Помимо описанных выше программ инициализации модуля E6udpmod_init_module и диспетчера условных системных вызовов 324 с именем mod_sys_E6call, модуль E6udpmod.ko содержит следующие программы: процедуры реализации условных вызовов E6_sendmsg, E6_rcvmsg, E6_regport, E6_unregport, E6_waitmsg; программу для обработки прибывших E6 пакетов E6_rcv; программу выхода из модуля (деинициализации) E6udpmod_cleanup_module. Взаимодействие указанных программ представлено на рис. 5.

Рассмотрим основные структуры данных модуля E6udpmod.ko. Заметим, что в соответствии с RFC 768 порт источника определенного приложения должен быть зарегистрирован. Сообщение может быть отправлено только с зарегистрированного порта так же, как сообщение может быть получено только на зарегистрированный порт. Таким образом, основной структурой данных является список зарегистрированных портов (E6 сокетов). Так как отправление сообщения полностью реализовано на протяжении исполнения одного системного вызова E6sendmsg, очереди выходных пакетов не создаются внутри модуля E6udpmod.ko; выделяется буфер пакета sk_buff, данные копируются в буфер из пользовательского адресного пространства, формируются E6 заголовки и, наконец, sk_buff размещается в очереди выходных пакетов посредством вызова функции драйвера hard_start_xmit.

При получении нового E6 пакета программа E6_rcv вызывается драйвером и пакет размещается в соответствующую очередь к зарегистрированному порту. Программа E6_rcv размещает пакет в хвост очереди, системный вызов E6_rcvmsg извлекает пакет из головы очереди (дисциплина FIFO). Если порт назначения не зарегистрирован (неизвестен), пакет теряется (уничтожается).

Таким образом, создаются два уровня списков: список зарегистрированных портов и списки (очереди) полученных пакетов (к зарегистрированным портам). Рис. 7 иллюстрирует взаимосвязи между основными структурами данных. Описание соответствующих структур данных задано следующим программным кодом:

```
struct e6portq {
	struct e6portq * next;
	struct e6portq * prev;
	u16 e6reg_port;
	pid_t pid;
```

```
        int qlen;
        struct sk_buf *skbhead;
        struct sk_buf *skbtail;
}

static struct e6portq * e6inpq_head = NULL;
static struct e6portq * e6inpq_tail = NULL;
```

Рассмотрим детально процесс отправления Е6 сообщений с помощью следующей программы:

```
int e6_sendmsg( struct e6msg_buf __user *msg )
{
        int len;
        struct sk_buff *skb;
        struct e6hdr *e6h;
        struct ethhdr *eth;
        unsigned long flags;
        int rc=0;

        copy_from_user(km, msg, sizeof(struct e6msg_buf)));
        len=km->len;
        if( e6find_regport( e6src_port ) == NULL )
        {
                rc = E6ERRUNREGPORT;
                return rc;
        }
        skb=alloc_skb(len+E6HEADSSPACE, GFP_KERNEL);
        skb->dev=e6_dev;
        skb->sk=NULL;

        /* Copy data */
        skb_reserve(skb, E6HEADSSPACE);
        copy_from_user( skb_put(skb,len), km->e6data, len );

        /* Build the E6P2 header. */
        skb_push(skb, sizeof(struct e6hdr));
        skb_reset_transport_header(skb);
        e6h = (struct e6hdr *)skb_transport_header(skb);
        e6h->e6dst_port = htons( km->e6dst_port );
        e6h->e6src_port = htons( km->e6src_port );
        skb_reset_network_header(skb);

        /* Build the E6Ethernet header */
        skb_push(skb, ETH_HLEN);
        skb_reset_mac_header(skb);
        eth = (struct ethhdr *)skb_mac_header(skb);
        skb->protocol = eth->h_proto = htons(ETH_P_E6);
        memcpy(eth->h_source, e6_myaddr, dev->addr_len);
        memcpy(eth->h_dest, km->e6dst_addr, dev->addr_len);

        /* Pass skb to the driver */
        local_irq_save(flags);
        netif_tx_lock(dev);
        rc=dev->hard_start_xmit(skb,dev);
        netif_tx_unlock(dev);
        local_irq_restore(flags);

        return rc;
}
```

Программа копирует заголовок из адресного пространства пользователя, проверяет, был ли зарегистрирован порт источника, выделяет буфер пакета skb, копирует данные из адресного пространства пользователя, заполняет заголовки e6h и eth и передаёт skb в драйвер. Рассмотрим краткое описание алгоритмов других программ модуля:

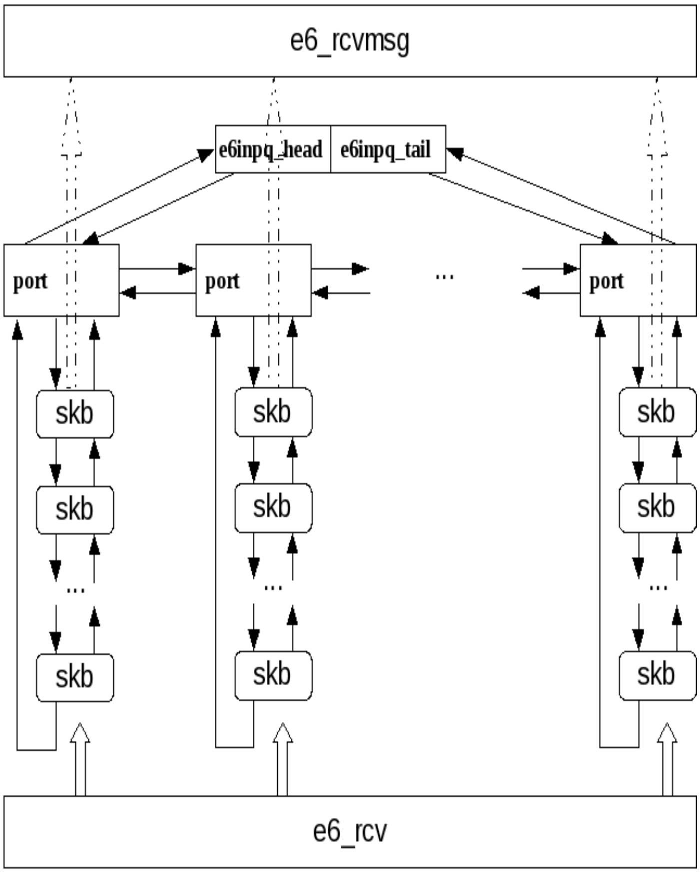


Рисунок 7 – Основные структуры данных

```
int e6_getmsg( u16 __user *upn, struct e6msg_buf __user *msg )
        // находит зарегистрированный порт с номером pn=*upn
        // проверяет, принадлежит ли порт pn текущему процессу
        // проверяет, пуста ли очередь skb к порту pn
        // извлекает skb из головы очереди
        // копирует заголовок сообщения и данные в адресное пространство пользователя
        // освобождает skb

int e6_rcv(struct sk_buff *skb, struct net_device *dev, struct packet_type *pt, struct net_device *orig_dev)
        // извлекает порт назначения dp из skb
        // находит зарегистрированный порт с номером dp; если отсутствует, теряет skb
        // проверяет ограничение длины очереди skb; если превышено, теряет skb
        // размещает skb в хвосте очереди к порту dp
        // проверяет флаг ожидания и пробуждает ждущий процесс

int e6_waitmsg(  u16 __user *upn )
        // checks weather the port pn=*upn is registered and belongs to the current process
        // checks the skb queue is empty; if empty, blocks the current processing
int e6_regport( u16 __user *upn )
        // проверяет, зарегистрирован ли порт pn=*upn; если да, возвращает ошибку
        // создаёт новую запись e6portq, заполняет её и размещает в списке зарегистрированных портов

int e6_unregport( u16 __user *upn )
        // проверяет, зарегистрирован ли порт pn=*upn; если нет, возвращает ошибку
        // проверяет, принадлежит ли порт pn текущему процессу; если нет, возвращает ошибку
        // освобождает все skb в очереди к порту pn (теряет непринятые пакеты)
```

```
    // извлекает запись e6portq из списка и освобождает её память
void e6udpmod_cleanup_module(void)
    // прекращает регистрацию типа пакетов e6: dev_remove_pack(&e6_packet_type);
    // прекращает регистрацию обработчика прерываний модуля: new_sys_e6call = NULL;
    // освобождает список портов и соответствующие очереди skb к портам
```

Заметим, что рассмотренные фрагменты кода достаточно упрощены для краткости изложения: опущены многочисленные проверки кодов возврата подпрограмм и соответствующие действия при возникновении ошибок.

Выводы

Таким образом, в настоящей статье представлена первая реализация нового стека протоколов E6 в среде ядра операционной системы. Выбрана операционная система Linux; использовано ядро версии 2.6.22.9. Работоспособность E6 сетей подтверждена успешным выполнением приложения E6talk среди других сетевых приложений и протоколов.

Реализован режим связи передачи дейтаграмм (аналог UDP). Реализация режима связи передачи сегментов данных с гарантированной доставкой информации (аналог TCP) на основе Ethernet LLC2 является направлением для будущих работ так же, как и создание шлюзов между E6 и TCP/IP сетями, которые в настоящее время существуют в «параллельных мирах».

Литература

1. Воробиенко П.П. Всемирная сеть Ethernet? / П.П. Воробиенко, Д.А. Зайцев, О.Л. Нечипорук // Зв'язок. – 2007. – № 5. – С. 14-19.
2. Воробієнко П.П. Спосіб передачі даних в мережі із заміщенням мережного та транспортного рівнів універсальною технологією канального рівня / П.П. Воробієнко, Д.А. Зайцев, К.Д. Гуляєв / Патент України на корисну модель № 35773, u2008 03069. – Заявл. 11.03.08; опубл. 10.10.08, Бюл. № 19.
3. Simulating E6 Protocol Networks using CPN Tools / [K.D. Guliaiev, D.A. Zaitsev, D.A. Litvin, E.V. Radchenko] // Proc. of Int. Conference on IT Promotion in Asia. – Tashkent (Uzbekistan), 2008. – P. 203-208.
4. Зайцев Д.А. Моделирование телекоммуникационных систем в CPN Tools / Зайцев Д.А., Шмелева Т.Р. – Одесса : ОНАС, 2009. – 72 с.
5. Postel J. Transmission control protocol / Postel J. // Information Sciences Institute: University of Southern California. – 1981, RFC 793. – 85 p.
6. Postel J. User Datagram Protocol / Postel J. // Information Sciences Institute: University of Southern California. – 1980, RFC 768. – 3 p.
7. IEEE Standard for Information technology-Telecommunications and information exchange between systems – Local and metropolitan area networks-Specific requirements. Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications. – LAN/MAN Standards Committee of the IEEE Computer Society, Approved 9 June 2005, IEEE-SA Standards Board IP. – 417 p.

К.Д. Гуляєв, Д.А. Зайцев

Експериментальна реалізація стека мережних протоколів Е6 в ядрі ОС Linux

Представлена перша реалізація нового стека мережних протоколів Е6 в ядрі операційної системи. Стек Е6 розроблено для підвищення ефективності роботи мереж цілком побудованих на основі технології Ethernet. Он використовує єдиний ієрархічний Е6 адрес на всіх рівнях еталонної моделі взаємодії відкритих систем и анулює протоколи TCP і IP. Експериментальна реалізація додає новий системний виклик ядра Linux и новий тип Ethernet E6 кадру. Усі стандарти прикладних інтерфейсів збережено згідно до RFC за виключенням використовування Е6 адреси замість IP-адреси, а також замість Ethernet MAC-адреси.

K.D. Guliaiev, D.A. Zaitsev

Experimental Realization of Networking Protocols Stack E6 within Linux Kernel

The first implementation of new E6 stack of networking protocols within a kernel of an operating system is presented. Stack E6 was developed to increase the efficiency of a network entirely built on the base of data-link Ethernet technology. It uses an uniform hierarchical E6 address on all the levels and annuls TCP and IP protocols. The experimental implementation adds a new system call to the kernel of Linux and a new type of Ethernet E6 frame. All the application interface standards are saved according to RFC except of E6 address usage instead of IP address and instead of Ethernet MAC address as well.

Статья поступила в редакцию 20.05.2009.